09/226,939

12/14/07

3

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G06F 15/403	A1	(11) International Publication Number: WO 92/15066 (43) International Publication Date: 3 September 1992 (03.09.92)

(21) International Application Number: PCT/US92/01458

(22) International Filing Date: 25 February 1992 (25.02.92)

(30) Priority data:
660,256 26 February 1991 (26.02.91) US

(71) Applicant: HEWLETT-PACKARD COMPANY [US/US]; 3000 Hanover Street, Palo Alto, CA 94304 (US).

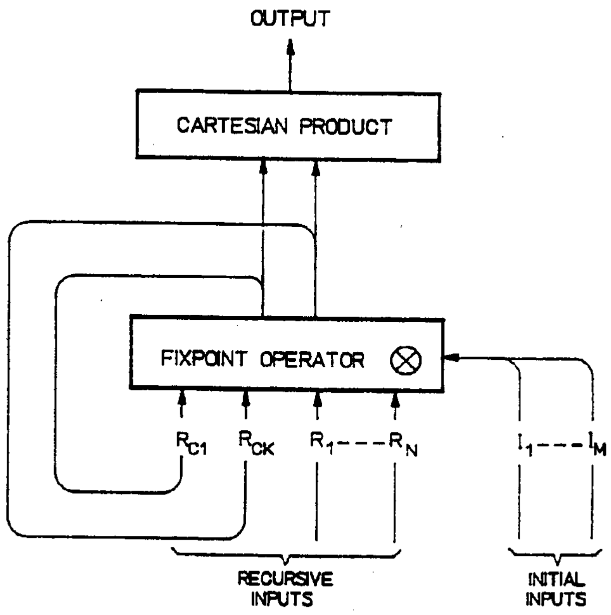
(72) Inventors: SHAN, Ming-Chien ; 13264 Glasgow Court, Saratoga, CA 95070 (US). NEIMAT, Marie-Anne ; 77 Magnolia Drive, Atherton, CA 94027 (US).

(74) Agents: SCHULZE, Herbert, R. et al.; Hewlett-Packard Company, Legal Dept., M/S 20B0, 3000 Hanover Street, Palo Alto, CA 94303 (US).

(81) Designated States: AT (European patent), BE (European patent), CH (European patent), DE (European patent), DK (European patent), ES (European patent), FR (European patent), GB (European patent), GR (European patent), IT (European patent), JP, LU (European patent), MC (European patent), NL (European patent), SE (European patent).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD OF EVALUATING A RECURSIVE QUERY OF A DATABASE



(57) **Abstract**

A method of evaluating a recursive query in a database (11). A recursive query is translated (13) into a relational algebra expression which includes a new fixpoint operator. The fixpoint operator supports mutually recursive and linearly recursive queries. The resulting expression is optimized (15) according to a set of fixpoint procedures.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	FI	Finland	ML	Mali
AU	Australia	FR	France	MN	Mongolia
BB	Barbados	GA	Gabon	MR	Mauritania
BE	Belgium	GB	United Kingdom	MW	Malawi
BF	Burkina Faso	GN	Guinea	NL	Netherlands
BG	Bulgaria	GR	Greece	NO	Norway
BJ	Benin	HU	Hungary	PL	Poland
BR	Brazil	IE	Ireland	RO	Romania
CA	Canada	IT	Italy	RU	Russian Federation
CF	Central African Republic	JP	Japan	SD	Sudan
CG	Congo	KP	Democratic People's Republic of Korea	SE	Sweden
CH	Switzerland			SN	Senegal
CI	Côte d'Ivoire	KR	Republic of Korea	SU	Soviet Union
CM	Cameroon	LI	Liechtenstein	TD	Chad
CS	Czechoslovakia	LK	Sri Lanka	TG	Togo
DE	Germany	LU	Luxembourg	US	United States of America
DK	Denmark	MC	Monaco		
ES	Spain	MG	Madagascar		

METHOD OF EVALUATING A RECURSIVE QUERY OF A DATABASE

CROSS-REFERENCE TO RELATED APPLICATION

This patent application is a continuation of patent application serial number 07/487,346, filed March 1, 1990, now abandoned, which in turn is a continuation-in-part of copending patent application serial number 07/286,425, filed December 19, 1988, and assigned to the same assignee as the present application.

BACKGROUND OF THE INVENTION

The present invention relates generally to database systems and more particularly to a method of evaluating a recursive query of a database.

Database systems are being used to store and manage more and more different kinds of data. As the use of database systems has expanded and the quantities of data stored in a database have increased, much effort has been devoted to improving existing database systems and developing new systems with new and better capabilities.

Relational Database Concepts

Data in a relational database are perceived by the user as being arranged in tables. Each table may be thought of as specifying a "relation" among the data in that table; therefore, each table is referred to as a "relation". Each row in a relation may be thought of as one data record. The rows are referred to as "tuples".

To "query" a database means to request information from it. To "evaluate" a query means to obtain the requested information. Sometimes the requested information can be obtained directly by looking it up in one of the relations. If the requested information does not appear in any of the relations, it must be derived, for example by comparing two or more tuples in a relation or by combining tuples from two different relations. The following four examples will help to illustrate these concepts.

EXAMPLE 1

Consider an historical database that contains data about people. Some of the data are arranged in a table or "relation" called PARENT. The data in the PARENT relation are arranged in the form

PARENT (NAME, PNAME)

where PNAME is the name of a parent of a person named NAME. For example, "PARENT (Andrew, William)" represents a tuple which says that William is a parent of Andrew.

Others of the data indicate who is a friend of whom; these data are arranged in a FRIEND relation in the form

FRIEND (NAME, FNAME)

where FNAME is the name of a friend of a person named NAME.

Still others of the data are in a PERSON relation and are arranged in the form

PERSON (NAME, SEX)

where NAME is the name of a person and SEX is the sex of that person. Representative PARENT, FRIEND and PERSON relations are depicted in Tables I through III, respectively.

PARENT relation

NAME (Name of Person)	PNAME (Name of Parent)
Andrew	William
Andrew	Mary
Mary	John
Mary	Anne
John	Richard
John	Wilma

TABLE I

FRIEND relation

NAME	FNAME
Andrew	Michael
Mary	Joan
William	Robert

TABLE II

PERSON relation

NAME	SEX
Andrew	M
William	M
Mary	F

TABLE III

A query of the form "FIND PARENTS OF [X]" is an example of a request for information that is directly obtainable by looking up data in this particular database. The response to such a query would be the names of the parents of X.

A query of the form "FIND MOTHER OF [X]" is an example of a request for information that cannot be obtained by looking it up. This is because the PARENT relation does not include the sex of the parents and, unlike a human, a computer does not know, for example, that "William" would ordinarily be a father and "Mary" a mother. Therefore, the requested information must be derived from information in the database, for example by (1) finding the parents of X in the PARENT relation, (2) finding the sex of each of those parents in the PERSON relation, and (3) selecting the female parent. The response to such a query would be the name of the mother of X.

EXAMPLE 2

Consider a factory database for keeping track of parts that are used to make engines. The data are arranged in two relations. The first relation, SUPPLIER, contains data arranged in the form

SUPPLIER (PART, SUPPLIER, CITY)

where PART represents the name (or part number or other convenient identifier) of a particular part, SUPPLIER represents a name of a source of that part, and CITY represents the location of that supplier. A few typical entries might be

SUPPLIER (Needle Valve, Bill's Brass Works, Buffalo)
SUPPLIER (Needle Valve, Paul's Plumbing Mfg. Co., Pittsburgh)
SUPPLIER (1/4" Screw, Mac's Machine Shop, Milwaukee)
SUPPLIER (Short Spring, Sam's Spring Specialties, Springfield)

and so on. The second relation, SUBPART, contains data in the form

SUBPART (SPART, SSUBPART, SQTY)

where SPART represents the name of a part that requires a subpart, SSUBPART represents the name of that subpart, and SQTY represents how many of that subpart are used in that part. Some examples are

SUBPART (Carburetor, Valve Assembly, 2)
SUBPART (Carburetor, 1/4" Screw, 16)
SUBPART (Carburetor, Short Spring, 3)
SUBPART (Valve Assembly, Needle Valve, 1)
SUBPART (Valve Assembly, 1/4" Screw, 4)

indicating that each carburetor requires two valve assemblies, sixteen 1/4" screws, three short springs, and so on.

A query of the form "FIND SUPPLIERS OF [X]" is an example of a request for information that is directly obtainable by retrieving data from this particular database. The response to such a query would be the names of all suppliers of part X.

A query of the form "HOW MANY SUPPLIERS SUPPLY [X]" is an example of a request for information that is not directly obtainable but that can be derived from information in the database, for example by finding all suppliers of X and then counting how many supplier names are found. The response to such a query would be the number of suppliers that supply part X.

EXAMPLE 3

Consider an airline reservation system. Data in this system are arranged in a FLIGHT relation in the form

FLIGHT (FROM, TO, DISTANCE, DTIME, ATIME, COST)

where FROM is the name of a departure city, TO is the name of an arrival city, DISTANCE is the mileage between those two cities, DTIME is the time of departure, ATIME is the time of arrival, and COST is the cost of a ticket on that flight. TABLE IV presents a typical FLIGHT relation.

FLIGHT relation

FROM	TO	DISTANCE	DTIME	ATIME	COST
Burbank	Reno	350	7:30 AM	8:40 AM	$ 250
Burbank	Reno	350	9:30 AM	10:40 AM	250
Burbank	Reno	350	11:30 PM	12:30 AM	150
Burbank	New York	2500	8:00 AM	4:00 PM	650
Reno	New York	2400	11:15 AM	7:30 PM	595

TABLE IV

A query of the form "FIND CHEAPEST FLIGHT BETWEEN [X] and [Y]" is an example of a request for information that cannot be obtained merely by retrieving data from this database but that can be derived from information in the database, for example by finding the fares of all flights between city X and city Y and then comparing the various fares to find which is lowest. The response to such a query would be the flight number of the cheapest flight between X and Y.

EXAMPLE 4

Consider a study of the spread of a sexually transmitted virus. One of the questions under investigation is the spread of the virus by heterosexual transmission. Records have been compiled of all heterosexual encounters in a defined population.

These records are arranged in an epidemic-study database in an ENCOUNTER relation in the form

ENCOUNTER (MALE, FEMALE, DATE, CITY, COUNTY).

Additional data respecting individuals in the population under study are arranged in a PERSON relation:

PERSON (PERSON, AGE, VACCINATED)

where VACCINATED is a yes-or-no entry indicating whether the person has been vaccinated against the virus. Representative ENCOUNTER and PERSON relations are depicted in Tables V and VI, respectively.

ENCOUNTER relation (E)				
e1	e2	e3	e4	e5
MALE	FEMALE	DATE	CITY	COUNTY
Andrew	Susan	12-2-88	Detroit	----
Andrew	Mary	9-6-88	New York	----
John	Susan	11-19-88	Boston	----
Richard	Susan	2-14-89	Boston	----
Joe	Anne	3-25-89	New York	----

TABLE V

PERSON relation (P)		
p1	p2	p3
PERSON	AGE	VACCINATED
Andrew	25	YES
John	18	NO
Richard	23	YES
Joe	40	YES
Susan	22	NO
Mary	29	YES
Anne	17	NO

TABLE VI

A query that requests the names of all women who had encounters with a certain man is an example of a request for information that can be retrieved directly. A query that seeks the names of all vaccinated women who had encounters with a certain man is an example of a request for information that can be derived by (1) retrieving the names of all women who had encounters with that man according to the ENCOUNTER relation and

(2) checking the names of each of those women according to the PERSON relation to find out which of them have been vaccinated.

Relational Algebra

As the volume of data in a database grows larger and the nature of the relations grows more complex, evaluating complicated queries becomes more difficult and time-consuming. Simplifying the task of evaluating a given query is known as "optimizing" the query. There may be only a few, or many hundreds, of ways to evaluate a complicated query, all of which provide the same answer but some of which are much more efficient than others. "Optimizing" such a query may mean choosing the best of all possible ways of evaluating the query, but usually "optimizing" means finding a reasonable number of ways according to techniques that are known to increase computational efficiency and choosing the best of those.

A set of relational operators collectively referred to as "relational algebra" has been developed for use in optimizing the evaluation of a complicated query in a large database. A query is translated into a relational algebra expression; the expression is simplified according to certain procedures; query plans for evaluating the simplified expression are generated; and the most efficient of these plans is selected and carried out to provide the desired response. See generally C. J. Date, *An Introduction to Database Systems* (4th Ed.) Vol. I, Addison-Wesley 1986, chapters 13 and 16, and references cited therein.

The relational algebra includes a set of operators. These operators can be compared with arithmetic operators such as "+" and "÷". Just as an arithmetic operator operates on one or two "input" numbers and provides a new "output" number, so each relational algebra operator takes one or two relations as "inputs" and provides a new relation as "output". A few examples of these operators are the SELECT, PROJECT, UNION, INTERSECTION, and JOIN operators.

The SELECT operator obtains specified rows from a relation. For example, a query seeking the names of Andrew's parents could be phrased as "SELECT entries respecting Andrew from the PARENT relation" (see Example 1 above). This SELECT operation would be represented in the relational algebra as

$$\sigma_{name="Andrew"}(PARENT).$$

In response, the SELECT operator would provide the following new relation:

Andrew	William
Andrew	Mary

from the PARENT relation.

The PROJECT operator obtains specified columns from a relation. For example, a query seeking the names of all persons who have children could be phrased as "PROJECT PNAME entries from the PARENT relation". This PROJECT operation would be represented in the relational algebra as

$$\pi_{pname}(PARENT)$$

and would provide the following new relation:

William
Mary
John
Anne
Richard
Wilma

from the PARENT relation.

The UNION operator collects all the rows of each of two relations. The UNION operation is represented in relational algebra as

$$A \cup B$$

where A and B are relations. For example, PARENT ∪ FRIEND in the historical database would provide a new relation containing parent-names and friend-names of everyone who has either a parent or a friend or both. In providing the new relation, the NAME entries in the PARENT relation would be correlated with the NAME entries in the FRIEND relation; for example, the name Andrew in the PARENT relation would be considered to refer to the same person as the name Andrew in the FRIEND relation.

The INTERSECTION operator collects common rows of each of two relations. The INTERSECTION operation is represented in relational algebra as

$A \cap B.$

For example, PARENT $\cap$ FRIEND in the historical database would provide a new relation containing parent-names and friend-names of those persons who have both a parent and a friend.

The JOIN operator combines rows from each of two relations according to a specified condition. In relational algebra the JOIN operation is represented as

$A \bowtie_{condition} B$

For example, the join operation $E \bowtie_{female=person} P$ in the epidemic-study database of Example 4 above would provide a new relation by combining the information in each tuple of the ENCOUNTER relation with the information in that tuple of the PERSON relation having an entry under PERSON that matches the entry under FEMALE in the tuple of the ENCOUNTER relation.

Recursive Queries

A kind of database query which has grown more important in recent years is a recursive query. Such a query can be described as a query which queries itself. A recursive query can be evaluated only by deriving information recursively. A general discussion of the mathematical concept of recursion can be found in J. Bradley, *Introduction to Discrete Mathematics*, ch. 6, Addison-Wesley 1988; see also E. Roberts, *Thinking Recursively*, John Wiley 1986.

As a simple example of a recursive query, consider a query of the form "FIND GRANDPARENTS OF [X]" directed to the historical database of example 1 above. This database contains no information about grandparents. However, the requested information can be recursively derived from the information in the database, for example by a query of the form "FIND PARENTS OF [FIND PARENTS OF [X]]".

If the number of iterations required to evaluate a recursive query is known in advance, then the evaluation process is relatively straight-forward. For example, the request to find the grandparents of X requires exactly two iterations -- one to find the parents of X and one to find the parents of the parents of X. However, if the number of iterations is not known, then the evaluation becomes far more difficult; an example of a request in which the number of iterations is not known is a request to find all ancestors of X.

As the volume of data in a database grows larger and the nature of the relations expressed by the data grows more complex, the time required for even a very powerful computer to respond to a complicated recursive query can become unacceptably long, especially when the number of iterations required to derive the response is not known in advance. Accordingly, the efficient evaluation of recursive queries has become a matter of critical importance in the design of modern database systems. A comprehensive survey of this problem is presented by F. Bancilhon and R. Ramakrishnan in "An Amateur's Introduction to Recursive Query Processing Strategies" in the *Proceedings of the ACM-SIGMOD Conference*, Washington, D.C., May 1986.

The relational algebra does not have recursion operators and hence cannot support recursive queries. Some relatively simple recursive queries can be expressed in transitive closure form, and transitive closure operators have been proposed for use in translating such queries into relational algebra expressions (R. Agrawal, "Alpha: An Extension of Relational Algebra to Express a Class of Recursive Queries", *Proceedings of the Third International Conference on Data Engineering*, Los Angeles, California, February 3-5, 1987; S. Ceri et al., "Translation and Optimization of Logic Queries: the Algebraic Approach", *Proceedings of the Eleventh International Conference on Very Large Data Bases*, Kyoto, Japan, August 1986). However, not all recursive queries can be expressed in transitive closure form.

From the foregoing, it will be apparent that there is a need for a way to optimize recursive queries, especially those which cannot be expressed in transitive closure form, for efficient evaluation in large and complex database systems.

SUMMARY OF THE INVENTION

The present invention provides a method of evaluating a recursive query in a database system by translating the query into an expression that includes a novel fixpoint operator and using a novel set of transformation procedures to simplify the translated query.

Briefly and in general terms, a method of evaluating a recursive query includes the steps of translating the query into a relational algebra expression that includes a fixpoint operator, optimizing the expression according to a set of transformation procedures, and evaluating the optimized expression by reference to data in the database. The transformation procedures include commuting a projection operation with a fixpoint operation, commuting a selection operation with a fixpoint operation, distributing a join operation over a fixpoint operation, and regrouping a join operation and a fixpoint operation.

Regrouping means applying the commutation and association rules, typically to an expression having a fixpoint and several join operators. The selection operation may be a selection predicate on a direct mapping column, a global selection predicate, or a selection predicate that includes a join operation.

Other aspects and advantages of the present invention will become apparent from the following detailed description, taken in conjunction with the accompanying drawings, illustrating by way of example the principles of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 is a data flow diagram depicting a preferred embodiment of a method of optimizing recursive queries according to the invention;

FIG. 2 is a flow diagram depicting initial and recursive inputs of a fixpoint operator as referenced in the "translate using ⊗" process of FIG. 1; and

FIG. 3 is a flow diagram depicting a generalized version of the fixpoint operator shown in FIG. 2.

DESCRIPTION OF THE PREFERRED EMBODIMENT

As shown in the drawings for purposes of illustration, the invention is embodied in a novel method of evaluating recursive queries in a database system. Relational algebra provides a powerful technique for optimizing the evaluation of database queries, but recursive queries have not been amenable to such optimization techniques.

In accordance with the invention, a novel fixpoint operator and new transformation procedures are provided for translating a recursive query into a relational algebra expression and then simplifying that expression. In the form of this simplified expression, the query can be evaluated much more efficiently than would otherwise be possible.

As shown in state diagram form in FIGURE 1, a method of evaluating a recursive query of a database 11 comprises translating a recursive query into an expression that includes a fixpoint operator, as indicated by a "translate using ⊗" process circle 13; optimizing the expression according to a set of transformation procedures as indicated by an "optimize" process circle 15; and evaluating the optimized expression, as indicated by an "evaluate" process circle 17, by reference to data in the database 11.

The recursive query is received from a user as indicated by an input box 19 and an arrow extending from the box 19 to the "translate" circle 13. The "user" may be a person at a computer terminal, but the user could also be, for example, an electronic device, an application program, or the like. Similarly, the result of evaluating the optimized query is provided to the user as indicated by an output box 21 and an arrow extending from the "evaluate" circle 17 to the box 21. The user who receives the output is usually the same user as the one who generated the query, but this need not be the case; the user that generates a query could specify that the result be sent somewhere else.

Several novel transformation procedures are provided according to the invention. These new transformation procedures include commuting a projection operation with a fixpoint operation as indicated by a "commute projection" process circle 25, commuting a selection operation with a fixpoint operation as indicated by a "commute selection" process circle 27, distributing a join operation over a fixpoint operation as indicated by a "distribute join" process circle 29, and regrouping join and fixpoint operations as indicated by a "regroup" process circle 31.

Arrows extend in both directions between the "optimize" process circle 15 and each of the process circles 25, 27, 29 and 31. These arrows indicate that application of one transformation procedure may result in an expression that requires application of another transformation procedure and that the optimization process may require one or more than one application of any given transformation procedure during the course of simplifying a given expression. Thus, depending on the characteristics of the expression being optimized, various ones of the transformation procedures may be used at various times during the optimization. Of course, some of the procedures may not be used at all in a given case.

Commuting a selection operation with a fixpoint operation comprises commuting a selection predicate on a direct mapping column with a fixpoint operation as indicated by a "direct map" process circle 33, commuting a global selection predicate with a fixpoint operation as indicated by a "global" process circle 35, and commuting a selection predicate that includes a join with a fixpoint operation as indicated by a "join" process circle 37. Arrows extend in both directions between the "commute selection" circle 27 and each of the circles 33, 35 and 37 to indicate that one or more than one of the procedures respecting commuting a selection and a fixpoint operator may be used as needed; of course, none of the "commute selection" procedures may be required in some cases.

Regrouping a join operation and a fixpoint operation comprises commutation and association as indicated by a "commutation" process circle 39 and an "association" process circle 41, respectively. Arrows extending in both directions between the "regroup" process circle 31 and each of the circles 39 and 41 indicate that one or more of the commutation and association procedures may be used. Again, some cases might not require either procedure.

In addition to the novel transformation procedures provided by the invention, a number of transformation procedures are already known in the relational algebra. Various ones of these may also be used to simplify a recursive query as indicated by a "transform procedures" process circle 43. Arrows extending in both directions between the circle 43 and the circle 15 indicate that these previously-known transformation procedures may be used once, several times or not at all in any given optimization.

The fixpoint operator according to the invention enhances the declarative power of relational algebra by supporting recursive queries. It is expected that the introduction of the fixpoint operator will benefit many database and computer applications such as computer aided design and manufacture (CAD/CAM), software engineering (CASE), and artificial intelligence (AI) applications.

As with other relational operators, the inputs (operands) and output of the fixpoint operator are relations. The fixpoint operator supports least fixed point semantics. The fixpoint operator can compute both linear and mutually recursive relations.

In a simple form the fixpoint operator is defined symbolically by an expression of the form

$$(I_1 - I_M) \otimes_{CE:OE} (R_1 - R_N , R_C)$$

where

I_j represents the *j*-th initial input,
CE represents a Condition Expression,
OE represents an Output Expression, and
R_j represents the *j*-th recursive input.

This form of the fixpoint operator is depicted diagrammatically in FIG. 2. Each initial input I is a relation. There may be one or more such inputs; *M* initial inputs are shown in FIG. 2. Each recursive input R is also a relation. There may be none, one or more of these recursive inputs; *N* recursive inputs are shown in FIG. 2. A recursive input may, but need not, be the same as an initial input. The output of the fixpoint operator is fed back as a recursive input R_C ; this recursive input R_C differs from the *N* other recursive inputs in that the input R_C is derived by the fixpoint operator whereas the other recursive inputs are not.

More particularly, during a first iteration the initial inputs are utilized to provide a first output. This output is fed back as the recursive input R_C . The recursive input R_C and the *N* other recursive inputs are utilized to provide a second output, and so on for as many iterations as are required.

If an output is not fed back as one of the recursive inputs, the fixpoint operator simplifies to conventional join and union operations.

In general, equality predicates in the condition expression are treated as join columns during processing of the ⊗ operator while any other predicates in the condition expression represent additional conditions which the tuples of the recursively derived relation generated by the ⊗ operator must satisfy.

A more general form of the fixpoint operator, corresponding to a plurality of *K* mutually recursive relations, is defined symbolically by an expression of the form

$$(I_1 - I_M) \otimes_{CE_1:OE_1, \ldots, CE_J:OE_J} (R_1 - R_N , R_{C1} - R_{CK}).$$

This form of the fixpoint operator is depicted diagrammatically in FIG. 3. Each initial input I is a relation. There may be one or more such inputs; *M* initial inputs are shown in FIG. 3. Each recursive input R is also a relation. There may be none, one or more of these recursive inputs; *N* recursive inputs are shown in FIG. 2. A recursive input may, but need not, be the same as an initial input. The fixpoint operator provides *K* recursive relations as outputs and these are fed back as recursive inputs R_{C1} through R_{CK} ; these *K* recursive inputs R_C differ from the *N* other recursive inputs in that the inputs R_C are derived by the fixpoint operator whereas the other recursive inputs are not. In general, *J* may but need not be equal to *K*.

The sets of initial or recursive inputs for each recursive relation need not be disjoint. For mutually recursive relations the recursive inputs cannot be disjoint because if they were the relations would not be mutually recursive.

The ultimate output of the ⊗ operator is a single relation representing the Cartesion product of the *K* mutually recursive output relations. Additional relational operators can be used to extract individual relations from the output.

The invention will now be more formally described in mathematical terms. The following examples will show how the fixpoint operator can be freely intermixed with other relational algebra operators to pose powerful queries.

The historical "people" database as described in Example 1 above includes the base relations PARENT, FRIEND and PERSON as given in Tables I through III. A derived relation ANCESTOR, of the form

ANCESTOR (DNAME, ANAME)

where DNAME is the name of a person and ANAME is the name of an ancestor of that person, is defined as

$$PARENT \otimes_{name=aname:\ dname,\ pname} (PARENT, ANCESTOR).$$

The fixpoint operator may then be used to express various recursive queries of the "people" database. Specifically, a query of the form "Find all ancestors of John" is expressed:

$$\pi_{aname}(\sigma_{dname=\text{"John"}}(PARENT \otimes_{name=aname:\ dname,\ pname} (PARENT, ANCESTOR)))$$

A query of the form "Find all people who are either friends of ancestors of John or friends of John" is expressed:

$$\pi_{fname}(\sigma_{dname=\text{"John"}}((PARENT \otimes_{name=aname:\ dname,\ pname} (PARENT, ANCESTOR)) \bowtie_{aname=name} FRIEND)) \cup \pi_{fname}(\sigma_{name=\text{"John"}} (FRIEND))$$

A query of the form "Find the friends of John's ancestors" is expressed

$$\pi_{fname}(\sigma_{dname=\text{"John"}}((PARENT \otimes_{name=aname:\ dname,\ pname} (PARENT, ANCESTOR)) \bowtie_{aname=name} FRIEND))$$

The factory database of Example 2 above includes the base relations SUPPLIER and SUBPART. A derived relation COMP, of the form

COMP (CPART, CSUBPART, CQTY),

is defined as

$$SUBPART \otimes_{spart=csubpart:\ ssubpart,\ cqty*sqty} (SUBPART, COMP).$$

The fixpoint operator may then be used to express various recursive queries such as a query of the form "Find the location and quantities of any parts that go into making a Locomotive" as follows:

$$\pi_{csubpart,city,sum(cqty)} (SUPPLY \bowtie_{part=csubpart} (GROUP\_BY_{csubpart:csubpart,sum(cqty)} (\sigma_{cpart=\text{"Locomotive"}}(SUBPART \otimes_{spart=csubpart:\ cpart,\ ssubpart,\ cqty*sqty} (SUBPART, COMP)))))$$

wherein the subscripts in the GROUP_BY operator indicate the group_by column and output columns of the GROUP_BY operator.

Similarly, in the airline reservation database of Example 3, a derived relation SHORT_CONNECTION of the form

SHORT_CONNECTION(START,DESTINATION,CDTIME,CATIME,CCOST),

is defined as

$$FLIGHT \otimes_{destination=from,\ catime\ <\ dtime,\ distance\ <\ 1000:\ start,\ to,\ cdtime,\ atime,\ ccost,\ +cost} (\ FLIGHT,\ SHORT\_CONNECTION\ )$$

The fixpoint operator is used, for example, to express a query of the form "Find the minimum cost of all flights between London and San Francisco under the condition that the distance between each pair of connecting points is less than 1,000 miles" as follows:

$$Min_{cost}\ (\sigma_{start="SF",\ destination="London"}\ (\sigma_{distance\ <\ 1000}\ (\ FLIGHT\ )$$
$$\otimes_{destination=from,\ catime\ <\ dtime,\ distance\ <\ 1000:\ start,\ to,\ cdtime,\ atime,\ ccost\ +\ cost}(\ FLIGHT,SHORT\_CONNECTION\ )$$

A main task to be performed by a query optimizer is to rearrange the sequence of operations in an expression of a query for more efficient evaluation. Starting with an initial form generated by a parser, the query expression usually undergoes a sequence of transformations based upon certain heuristic rules or execution cost comparisons. The transformations usually include:

Performing selections and projections as early as possible,

Combining sequences of the same operation (e.g. selection or projection) into one operation,

Commuting selection or projection operations with join or Cartesian product operations,

Commuting join (or Cartesian product) operations, and

Re-associating join (or Cartesian product) operations.

(See, generally, Ullman, J., *Principles of Database Systems* (2nd ed.), Computer Science Press, Maryland, 1982 for more discussion of transformations).

These transformations are made possible by properties (such as commutativity and associativity) which are inherent in these operators. In order to avoid the development of a new paradigm for dealing with recursive queries, the present invention stays within the realm of relational algebra so that existing efficient implementation algorithms and other useful techniques (e.g. optimal plan search mechanisms) can be employed. However, not all of the common algebraic properties are held between fixpoint operators and other relational algebra operators. To integrate the fixpoint operator into an existing query optimization process, the valid transformation rules in the extended relational algebra have to be identified.

Example 4 above (the study of the sexually transmitted virus) will be used to illustrate these rules.

Assume that one of the questions under investigation is the spread of the virus by heterosexual transmission. To this end, a record has been kept of all heterosexual encounters in a population under study. The ENCNTR relation is used to store these records. The relation contains additional information about each person in the population under study. Now consider the two derived relations EXPSDFML (MALE, FEMALE, DATE, CITY, COUNTY) and EXPSDML (FEMALE, MALE, DATE, CITY, COUNTY). The relation EXPSDFML is defined as:

$$\pi_E(([\pi_{e1,\, e2,\, e3,\, e4,\, e5}E], [\pi_{e2,\, e3,\, e4,\, e5}E]) \otimes_{[f2=m1,\, f3<m3,\, m2=e1,\, m3<e3\, :\, f1,\, e2,\, e3,\, e4,\, e5],\ [m2=f1,\, m3<f3,\, f2=e2,\, f3<e3\, :\, m1,\, e1,\, e3,\, e4,\, e5]}((E,F,M), (E,F,M)))$$

The projection is needed since the output of the fixpoint operator is, by definition, the cross product of relations F and M. There is no need to compute the cross product of F and M in such a case, and the query processor will detect that. Similarly, the relation EXPSDML is defined as:

$$\pi_F(([\pi_{e1,\, e2,\, e3,\, e4,\, e5}E], [\pi_{e2,\, e1,\, e3,\, e4,\, e5}E])$$

$\otimes_{[f2=m1,\ f3<m3,\ m2=e1,\ m3<e3\ :\ f1,\ e2,\ e3,\ e4,\ e5],\ [m2=f1,\ m3<f3,\ f2=e2,\ f3<e3\ :\ m1,\ e1,\ e3,\ e4,\ e5]}((E,F,M),\ (E,F,M)))$

Tables V and VI above illustrate the ENCOUNTER(E) and PERSON(P) relations. The EXPSDFML(F) AND EXPSDML(M) relations are illustrated in the following tables VII and VIII.

EXPSDFML relation (F)

f1	f2	f3	f4	f5
MALE	FEMALE	DATE	CITY	COUNTY
—	—	—	—	—

TABLE VII

EXPSDML relation (M)

m1	m2	m3	m4	m5
FEMALE	MALE	DATE	CITY	COUNTY
—	—	—	—	—

TABLE VIII

Now, consider query Q1 which finds all vaccinated females who might have been exposed to the virus either directly or indirectly through make carrier X and such that all encounters leading to each female have taken place in New York. The query is expresses in the extended relational algebra as follows:

$$\pi_{f2}(\sigma_{f1="X",\ G:f4="NY",\ G:\ m4="NY",\ p3=true}(P\bowtie_{p1=f2}(([\pi_{e1,\ e2,\ e3,\ e4,\ e5}E],\ [\pi_{e2,\ e1,\ e3,\ e4,\ e5}E])\ \otimes_{[f2=m1,\ f3<m3,\ m2=e1,\ m3<e3\ :\ f1,\ e2,\ e3,\ e4,\ e5],\ [m2=f1,\ m3<f3,\ f2=e2,\ f3<e3\ :\ m1,\ e1,\ e3,\ e4,\ e5]}((E,F,M),\ (E,F,M)))))$$

Note that a "G" tag attached to a selection condition indicates that it is a global one. A selection predicate is said to be global if it is applied at each iteration during the generation of the recursive relation defined by the fixpoint operation. In other words, once a tuple of an input relation fails to satisfy a global selection predicate, it will be excluded from consideration in any of the subsequent recursive computations. (The detection of global predicates will be discussed later.)

In general, for queries on recursive relations, a detailed semantic analysis is required to determine the legal transformations that can be applied to an algebraic expression. The present discussion is limited to transformation rules which do not require any semantic query analysis. Formal proofs of the rules are omitted because of their length and instead each rule is motivated either with a sketch of a proof or with detailed examples.

1. Commuting projection operation with fixpoint operation

Rule 1: Let $IINP_i$ and $RINP_i$ be the initial and recursive input expressions for mutually recursive relation R_i of the fixpoint operation. PE_i denotes the set of attributes of R_i in the projection operation. SE_i denotes the set of attributes of R_i in the selection operation. CE_i and OE_i denote the sets of attributes in the condition expression and output expression of R_i in the fixpoint operation. $A(IINP_i)$ denotes the set of attributes in the initial input expression and $A(RINP_i)$ the set of attributes in the recursive input expression for relation R_i. Then

$$\pi_{PE_1, \ldots, PE_n}(\sigma_{SE_1, \ldots, SE_n}((IINP_1, \ldots, IINP_n) \otimes_{CE_1 : OE_1, \ldots, CE_n : OE_n} (RINP_1, \ldots, RiNP_n))) \equiv \pi_{PE_1, \ldots, PE_n}(\sigma_{SE_1, \ldots, SE_n}((\pi_{PE'_1} IINP_1, \ldots, \pi_{PE'_n} IINP_n) \otimes_{CE_1 : OE'_1, \ldots, CE_n : OE'_n} (\pi_{PE''_1} RINP_1, \ldots, \pi_{PE''_n} RINP_n)))$$

where

$$PE_i' = (PE_i \cup SE_i \cup (\cup_{j=1}^{n} CE_j)) \cap A(IINP_i),$$

$$PE_i'' = (PE_i \cup SE_i \cup (\cup_{j=1}^{n} CE_j)) \cap A(RINP_i), \text{ and}$$

$$OE_i' = (PE_i \cup SE_i \cup (\cup_{j=1}^{n} CE_j)) \cap OE_i$$

Sketch of proof: In principle, values for columns not required in any subsequent operations can be discarded. Therefore, during the process of inputs for the fixpoint operation, only the values for columns being referenced in the condition or output expressions of the fixpoint operator or in other subsequent operations need to be retrieved. For example, using query Q1, this rule can be applied to transform it as follows:

$$\pi_{f2}(\sigma_{f1=\text{"X"},\ G:f4=\text{"NY"},\ G:m4=\text{"NY"},\ p3=true}\,(P \bowtie_{p1=f2} (([\pi_{e1,\ e2,\ e3,\ e4,\ e5}E], [\pi_{e2,\ e1,\ e3,\ e4,\ e5}E]) \otimes_{[f2=m1,\ f3<m3,\ m2=e1,\ m3<e3\ :\ f1,\ e2,\ e3,\ e4,\ e5],\ [m2=f1,\ m3<f3,\ f2=e2,\ f3<e3\ :\ m1,\ e1,\ e3,\ e4,\ e5]} ((E,F,M,), (E,F,M))))) \equiv$$

$$\pi_{f2}(\sigma_{f1=\text{"X"},\ G:f4=\text{"NY"},\ G:m4=\text{"NY"},\ p3=true}\,(P \bowtie_{p1=f2} (([\pi_{e1,\ e2,\ e3,\ e4}E], [\pi_{e2,\ e1,\ e3,\ e4}E]) \otimes_{[f2=m1,\ f3<m3,\ m2=e1,\ m3<e3\ :\ f1,\ e2,\ e3,\ e4],\ [m2=f1,\ m3<f3,\ f2=e2,\ f3<e3\ :\ m1,\ e1,\ e3,\ e4]} ((\pi_{e1,\ e2,\ e3,\ e4}E, \pi_{f1,\ f2,\ f3,\ f4}F, \pi_{m1,\ m2,\ m3,\ m4}M), (\pi_{e1,\ e2,\ e3,\ e4}E, \pi_{f1,\ f2,\ f3,\ f4}F, \pi_{m1,\ m2,\ m3,\ m4}M)))))$$

2. *Commuting selection operations with fixpoint operations.*

This discussion will explore the heuristic of "performing selection as early as possible". In general, it means to move the selections inside other operators as far as possible. For query Q1, all of its selections can be applied to base relations E and P directly rather than the final result composed by the join operation and fixpoint operation. The original expression is then translated into:

$$\pi_{f2}((\sigma_{p3=true}P) \bowtie_{p1=f2} (([\sigma_{e1=\text{"X"},\ e4=\text{"NY"}}(\pi_{e1,\ e2,\ e3,\ e4,\ e5}E)] [\sigma_{e4=\text{"NY"}}(\pi_{e2,\ e1,\ e3,\ e4,\ e5}E)]) \otimes_{[f2=m1,\ f3<m3,\ m2=e1,\ m3<e3\ :\ f1,\ e2,\ e3,\ e4,\ e5],\ [m2=f1,\ m3<f3,\ f2=e2,\ f3<e3\ :\ m1,\ e1,\ e3,\ e4,\ e5]} (((\sigma_{e4=\text{"NY"}}E),F,M), ((\sigma_{e4=\text{"NY"}_7}E),F,M))))$$

In the above expression, the global selections (G:f4="NY",G:m4="NY") are converted to regular selections and applied to both initial and recursive inputs of the fixpoint operation. The purpose of this translation is to reduce the sizes of the operands for each operation. If the fixpoint operator ⊗ is viewed as a generator of a directed graph consisting of all paths leading to all possible answers, an early selection on the initial input and recursive input relations has the effect of eliminating the unqualified paths in the graph before they are generated. See, generally, Ioannidis, Y., and Wong, W., "On the Computation of the Transitive Closure of Relational Operators", *Proc. of*

12th Int. Conf. on VLDB, Tokyo, Japan, August, 1986; Jagadish, H., Agrawal, R., and Ness, L., "A Study of Transitive Closure as a Recursive Mechanism", *Proc. of ACM-SIGMOD 1987 Int. Conf. on Management of Data*, San Francisco, California, May 1987; and Lu, H., "New Strategies for Computing the Transitive Closure of a Database Relation", *Proc. of 13th. Int. Conf. on VLDB*, Brighton, England, September, 1987.

While the legality of moving global selections into the fixpoint operator will be apparent, the movement of the non-global selection ($\sigma_{f1=\text{"X"}}$) into the initial input needs some explanation. After a careful examination, it can be found that the two forms of the query are semantically equivalent. The is because the selection ($\sigma_{f1=\text{"X"}}$) is applied to a direct mapping column of the recursive relation EXPSDFML. A column of a recursive relation is considered a direct mapping column if the output expression for that column consists of that column itself only. The crucial characteristic of a direct mapping column is that is acquires its entire set of values from the initial input relation(s). The values it assumes during each subsequent recursive iteration are always taken from the value set of its own initial input and are not computed from values of other resources. Thus, once the set of values of the initial input is determined, no new values are added to the column. A careful look at the definition of EXPSDFML will show that new values are added to columns f_2, ..., f_5 during each recursive iteration but not to column f_1.

The concept of direct mapping column is very similar to that of invariant column introduced in Devanbu, P. and Agrawal, R., "Moving Selections Into Fixpoint Queries", *Proc. of 4th Int. Conf. on Data Engineering*, Los Angeles, February, 1988. Direct mapping columns are actually a subset of invariant columns. However, the detection of the more general invariant columns requires a detailed analysis of the selection predicates. The advantage of concentrating on direct mapping columns is that their detection is trivial and they cover the majority of the cases for which the payoff for performing early selections is substantial.

Selections which are neither global nor applied to direct mapping columns will now be considered. Let Q2 be the query to find all females who might have been exposed to the virus either directly or indirectly through male carrier X and such that the last encounter took place in New York. The query is expressed in the extended relational algebra as follows:

$$\pi_{f2}(\sigma_{f1=\text{"X"},\ f4=\text{"NY"}}(([\pi_{e1,\ e2,\ e3,\ e4,\ e5}E],\ [\pi_{e2,\ e1,\ e3,\ e4,\ e5}E]) \otimes_{[f2=m1,\ f3<m3,\ m2=e1,\ m3<e3\ :\ f1,\ e2,\ e3,\ e4,\ e5\ ],\ [m2=f1,\ m3<f3,\ f2=e2,\ f3<e3\ :\ m1,\ e1,\ e3,\ e4,\ e5\ ]}((E,F,M),\ (E,F,M))))$$

Distributing the selection ($\sigma_{f4=\text{"NY"}s}$) over the fixpoint operator would result in:

$$\pi_{f2}(\sigma_{f1=\text{"X"}}(([(\sigma_{e4=\text{"NY"}}(\pi_{e1,\ e2,\ e3,\ e4,\ e5}E\ ))],[\sigma_{e4=\text{"NY"}}(\pi_{e2,\ e1,\ e3,\ e4,\ e5}E)]) \otimes_{[f2=m1,\ f3<m3,\ m2=e1,\ m3<e3\ :\ f1,\ e2,\ e3,\ e4,\ e5],\ [m2=f1,\ m3<f3,\ f2=e2,\ f3<e3\ :\ m1,\ e1,\ e3,\ e4,\ e5\ ]}(((\sigma_{e4=\text{"NY"}}E\ ),F,M),\ ((\sigma_{e4=\text{"NY"}}E\ ),F,M))))$$

However, the above query expression will not generate the complete set of answers. This is due to the exclusion of qualified "bridge tuples" from the intermediate results used for computing the EXPSDFML and EXPSDML relations. According to the above expressions, only those tuples satisfying ($\sigma_{e4=\text{"NY"}}E$) participate in the initial and recursive inputs to the relation EXPSDFML. In order to produce the complete set of answers, all qualified "bridge tuples" need to be included. It means that all tuples which do not satisfy ($\sigma_{e4=\text{"NY"}}\ E$) must still be saved for subsequent computation to avoid the loss of certain answers.

The rules governing the movement of selections into fixpoint operators thus depend on whether the selections are global, and if not global whether they apply to direct mapping columns. They are formulated as follows:

A. Commuting selection predicates on direct mapping columns with fixpoint operations.

Rule 2: Let SED_i be the set of selection predicates on direct mapping columns for mutually recursive relation R_i, where SED_i contains no reference to any column of mutually recursive relation R_j where $i \neq j$. Then

$$\sigma_{SED_1, \ldots, SED_n}((IINP_1, \ldots, IINP_n) \otimes_{CE_1 : OE_1, \ldots, CE_n : OE_n} (RINP_1, \ldots, RINP_n)) \equiv (\sigma_{SED_1} IINP_1, \ldots, \sigma_{SED_n} IINP_n) \otimes_{CE_1 : OE_1, \ldots, CE_n : OE_n} (RINP_1, \ldots, RINP_n))$$

For example, as seen earlier, this rule was used to move the selection predicate $(\sigma_{f1 = "X"})$ into input relation E for query Q1.

B. Commuting global selection predicates with fixpoint operations

Rule 3: Let SEG_i be the set of all global predicates in the selection expression referencing attributes of mutually recursive relation R_i where SEG_i contains no reference to any column of mutually recursive relation R_j where $i \neq j$; and let SEG_i' be the same set of predicates after the "G" tag has been removed. Then

$$\sigma_{SEG_1, \ldots, SEG_n}((IINP_1, \ldots, IINP_n) \otimes_{CE_1 : OE_1, \ldots, CE_n : OE_n} (RINP_1, \ldots, RINP_n)) \equiv (\sigma_{SEG_1'} IINP_1, \ldots, \sigma_{SEG_n'} IINP_n) \otimes_{CE_1 : OE_1, \ldots, CE_n : OE_n} (\sigma_{SEG_1'} RINP_1, \ldots, \sigma_{SEG_n'} RINP_n))$$

For example, as seen earlier, this rule was used to move the global selection predicates (f_4 = "NY") and (m_4 = "NY") in query Q1 into the initial and recursive inputs of the fixpoint operator.

The designation of global predicates is left to the user (see, generally, Rosenthal, A., Heiler, S., Dayal, U., and Manola, F., "Traversal Recursion: A Practical Approach to Support Recursive Applications", *Proc. of ACM-SIGMOD 1986 Int. Conf. on Management of Data*, Washington, D.C., May 1986). As was exposed earlier, the detection of direct mapping columns is straightforward. The detection of such columns may be performed when a recursive relation is defined and that information permanently stored in the system catalogs.

Thus far, selections wherein all the columns involved belong to a single mutually recursive relation have been considered. Selections of the form $f_i \,\theta\, f_j$ where f_i is a column of mutually recursive relation R_i, f_j is a column of mutually recursive relation R_j where $i \neq j$, and θ is a comparison operator, will now be discussed.

Consider query Q3 which finds all males who might have been indirectly exposed to the virus through make carrier X. The query is expressed as follows:

$$\pi_{m2}\,(\sigma_{f1=\text{"X"},\, f2=m1}\,(([\pi_{e1,\, e2,\, e3,\, e4,\, e5}\, E\,],\ [\pi_{e2,\, e1,\, e3,\, e4,\, e5}\, E\,]\ \otimes_{[f2=m1,\, f3<m3,\, m2=e1,\, m3<e3\,:\, f1,\, e2,\, e3,\, e4,\, e5\,],\ [m2=f1,\, m3<f3,\, f2=e2,\, f3<e3\,:\, m1,\, e1,\, e3,\, e4,\, e5]}\, ((E,F,M),\ (E,F,M)))$$

The selection $f2 = m1$ is nothing but a join of the two recursive relations EXPSDFML and EXPSDML. This will be detected by the query optimizer and query Q3 will be translated to:

$$\pi_{m2}\,(\sigma_{f1=\text{"X"}}\,(\pi_F\,(([\pi_{e1,\, e2,\, e3,\, e4,\, e5}\, E\,],\ [\pi_{e2,\, e1,\, e3,\, e4,\, e5}\, E\,])\ \otimes_{[f2=m1,\, f3<m3,\, m2=e1,\, m3<e3\,:\, f1,\, e2,\, e3,\, e4,\, e5\,],\ [m2=f1,\, m3<f3,\, f2=e2,\, f3<e3:\, m1,\, e1,\, e3,\, e4,\, e5]}\, ((E,F,M\,),(E,F,M\,)))\ \bowtie_{f2=m1}\ \pi_M\ (([\pi_{e1,e2,\, e3,\, e4,\, e5}\, E],\ [\pi_{e2,\, e1,\, e3,\, e4,\, e5}\, E\,])\ \otimes_{[f2=m1,\, f3<m3,\, m2=e1,\, m3<e3\,:\, f1,\, e2,\, e3,\, e4,\, e5\,],\ [m2=f1,\, m3<f3,\, f2=e2,\, f3<e3\,:\, m1,\, e1,\, e3,\, e4,\, e5]}\, ((E,F,M\,),\ (E,F,M\,)))))$$

At execution time, the fixpoint operation will be performed only once, generating both recursive relations EXPSDFML and EXPSDML simultaneously. A join will then be performed on both relations. That is, no cross products or projections of relations EXPSDFML and EXPSDML will actually take place to evaluate query Q3. This rule is formulated as:

C. Detection of join operations

Rule 4: Let $f_i \,\theta\, f_j$ be a selection where f_i is a column of mutually recursive relation R_i, f_j is a column of mutually recursive relation R_j where $i \neq j$, and θ is a comparison operator. Then

$$\sigma_{f_i \theta f_j}(([Inp_1], \ldots, [Inp_n]) \otimes_{[Cond_1 : Out_1], \ldots, [Cond_n : Out_n]} (Recinp_1, \ldots, Recinp_n)) \equiv \pi_{R_i}(([Inp_1], \ldots, [Inp_n]) \otimes_{[Cond_1 : Out_1], \ldots, [Cond_n : Out_n]} (Recinp_1, \ldots, Recinp_n)) \bowtie_{f_i \theta f_j} \pi_{R_j}(([Inp_1], \ldots, [Inp_n]) \otimes_{[Cond_1 : Out_1], \ldots, [Cond_n : Out_n]} (Recinp_1, \ldots, Recinp_n))$$

3. *Distributing join operations over fixpoint operations*

In traditional query optimization, selections are moved inside joins to reduce the size of the operands of the join operations. This is usually a good strategy because selections always reduce the sizes of their operands. On the other hand, the result of a join operation may be smaller or larger than the sizes of its operands. But the result of fixpoint operation is always larger than the size of its initial inputs. Therefore, a query optimizer should never consider moving a fixpoint operator inside a join operator, but should assess the value of moving a join operator inside a fixpoint operator.

Consider again query Q1. One can safely move the join operator (without the selection) inside the fixpoint operator. The query translates to:

$$\pi_{f2}(\sigma_{f1="X",\ G: f4="NY",\ G: m4="NY",\ p3=true}((([P \bowtie_{p1=e2} E], [E]) \otimes_{[f2=m1,\ f3<m3,\ m2=e1,\ m3<e3\ :\ f1,\ e2,\ e3,\ e4,\ e5,\ p2,\ p3],\ [m2=f1,\ m3<f3,\ f2=e2,\ f3<e3\ :\ m1,\ e1,\ e3,\ e4,\ e5]} ((P \bowtie_{p1=e2} E,F,M), (E,F,M))))$$

However, the distribution of join operators over fixpoint operators is not always as trivial. As an example, consider a slightly modified schema where the relation PERSON is replaced by the relation VACCINATED-PERSON(person, age) which contains the age of each vaccinated person in the population under study. The relation is depicted in the Table IX as:

VACCINATED PERSON relation (V)	
v1	v2
PERSON	AGE
----	----

TABLE IX

Now, let Q4 be the query to find all vaccinated females and their age if they have been exposed to the virus either directly or indirectly through male carrier X. The query is expressed as follows:

$$\pi_{f2,v2}(\sigma_{f1=\text{"X"}}(V \bowtie_{v1=f2}(([E], [E]) \otimes_{[f2=m1,\ f3<m3,\ m2=e1,\ m3<e3\ :\ f1,\ e2,\ e3,\ e4,\ e5],\ [m2=f1,\ m3<f3,\ f2=e2,\ f3<e3\ :\ m1,\ e1,\ e3,\ e4,\ e5]} ((E,F,M), (E,F,M)))))$$

Attempting to distribute the join operator results in:

$$\pi_{f2,v2}(\sigma_{f1=\text{"X"}}(([V \bowtie_{v1=e2} E], [E]) \otimes_{[f2=m1,\ f3<m3,\ m2=e1,\ m3<e3:\ f1,\ e2,\ e3,\ e4,\ e5,\ v2],\ [m2=f1,\ m3<f3,\ f2=e2,\ f3<e3\ :\ m1,\ e1,\ e3,\ e4, e5]} ((V \bowtie_{v1=e2} E,F,M), (E,F,M))))$$

However, this transformation will not produce the correct result. This is again due to the loss of qualified "bridge tuples". Although non-vaccinated females are not requested in query Q4, their presence during the iterative process is essential to finding all vaccinated females who have been exposed indirectly to the virus carried by "X". In order to preserve the correct answer, the join operator has to transform to a right outer join operator (see, generally, Date, C., *Relational Databases: Selected Writings*, Addison-Wesley Publishing Company, 1986) as it moves inside the fixpoint operator. During query execution, all tuples which are strictly the result of the outer join may be marked so they are eliminated from the final result.

If the join is over a direct mapping column, the join operator need not be converted to a right outer join operator. Consider the modified schema again and let Q5 be the query to find all vaccinated males, their ages, and all females who have been exposed to them directly or indirectly. The query is expressed as follows:

$$\pi_{f1,\ v2,\ f2}(V \bowtie_{v1=f1}(([E], [E]) \otimes_{[f2=m1,\ f3<m3,\ m2=e1,\ m3<e3\ :\ f1,\ e2,\ e3,\ e4,\ e5],\ [m2=f1,\ m3<f3,\ f2=e2,\ f3<e3\ :\ m1,\ e1,\ e3,\ e4,\ e5]} ((E,F,M), (E,F,M))))$$

The distribution of the join operator results in:

$$\pi_{f1,\ v2,\ f2}(([V \bowtie_{v1=e1} E],\ [E])\ \otimes_{[f2=m1,\ f3<m3,\ m2=e1,\ m3<e3\ :\ f1,\ e2,\ e3,\ e4,\ e5,\ v2],\ [m2=f1,\ m3<f3,\ f2=e2,\ f3<e3\ :\ m1,\ e1,\ e3,\ e4,\ e5]}\ ((V \bowtie_{v1=e1} E,F,M),\ (E,F,M)))$$

which will produce the correct answers because the join was over a direct mapping column.

For simplicity, the distribution of joins is performed over the initial inputs and non-recursive relations in the recursive inputs and only when the join columns are restricted to a single non-recursive relation in the inputs.

Rule 5: Let $IINP_i$ and $RINP_i$ be the initial and recursive input expressions for mutually recursive relation R_i of a fixpoint operation. Let REGEXP be a regular relational expression whose output is, as usual, a single relation. Consider the expression:

$$REGEXP \bowtie_{x\ \theta\ fj},\ ((IINP_1,\ \ldots,\ IINP_N)\ \otimes_{CE_1\ :\ OE_1,\ \ldots,\ CE_n\ :E_n}(RINP_1,\ \ldots,\ RINP_n))$$

where x is one of the columns of REGEXP, f_j is one of the columns of mutually recursive relation R_j, and θ is a comparison operator. Consider the recursive input expression $RINP_j$ for relation R_j. It consists of the recursive relations $R_1,\ \ldots,\ R_n$, and some non-recursive relations $NR_1,\ \ldots,\ NR_m$. Similarly, the initial input expression $IINP_j$ for relation R_j consists of relations $I_1,\ldots,I_l$. If column f_j takes all its initial inputs from a single column of a single input relation I_k and all its recursive inputs from a single column of a non-recursive relation NR_v, then the above expression is equivalent to:

$$(IINP_1,\ \ldots,\ IINP_{j-1},\ (I_1,\ \ldots,\ REGEXP \infty_{x\ \theta\ fj'} I_k,\ \ldots,\ I_l),\ IINP_{j+1},\ \ldots, IINP_n)\ \otimes_{CE_1\ :\ OE_1,\ \ldots,\ CE_n\ :\ OE_n}(RINP_1,\ \ldots,\ RINP_{j-1}, (NR_1,\ \ldots,\ REGEXP \infty_{x\ \theta\ fj''}\ NR_v,\ \ldots,\ NR_m,\ \ R_1,\ \ldots,\ R_n),\ RINP_{j+1},\ \ldots,\ RINP_n)$$

where ∞ indicates a right outer join or regular join operator, depending upon whether the join column f_j is a direct mapping column or not, and where $f_j{}'$ is the appropriate column in relation I_k and $f_j{}''$ is the appropriate column in relation NR_v.

4. Commuting and associating join operations

Rule 6: The commutative and associative algebraic laws for join operations still hold if each fixpoint operation is treated as a whole. That is,

$$A \bowtie (B \otimes (C,D)) \bowtie E \equiv A \bowtie E \bowtie (B \otimes (C,D)), \text{ and}$$

$$(A \bowtie B) \bowtie (C \otimes (D,E)) \equiv A \bowtie (B \bowtie (C \otimes (D,E)))$$

These commutative and associative properties allow the query optimizer to choose the best execution operations.

The algebraic properties of fixpoint operators developed above give the query optimizer more leeway in choosing efficient execution strategies. The transformation rules representing these properties can be added in a fairly straightforward manner to most existing query optimizers.

Most implementations of logic databases (see, for example, Morris, K., Ulman, J., and Gelder, A., "Design Overview of the NAIL System", *Proc. 3rd Int. Conf. on Logic Programming*, 1986, and Zaniolo, C., and Sacca, D., "Rule Rewriting Methods for Efficient Implementation of Horn Logic", *MCC Technical Report DB-084-87*, March 1987) do not rely on any statistical information to determine their execution strategy. They commonly use simple heuristics which choose to extend the predicate with the largest number of bound arguments. One of the most promising features of the present approach is that it is targeted at existing relational query optimizers. Thus, the transformation rules presented herein become a tool for the query optimizer to choose among a menu of execution strategies based upon the estimated execution costs associated with each form a query can take.

From the foregoing it will be appreciated that the invention provides an effective and efficient method of evaluating linear and recursive queries in large databases. Existing relational techniques are integrated with the novel fixpoint operator and transformation procedures provided by the invention to optimize even very complex recursive queries.

Although a specific embodiment of the invention has been described and illustrated, the invention is not to be limited to the specific forms or arrangements of parts so described and illustrated, and various modifications and changes can be made without departing from the scope and spirit of the invention. Within the scope of the appended claims, therefore, the invention may be practiced otherwise than as specifically described and illustrated.

CLAIMS

1. A method of evaluating a recursive query (19) of a computerized database (11), the method comprising: translating a recursive query into an expression that includes a fixpoint operator (13); optimizing the expression according to a set of transformation procedures (15); and evaluating the optimized expression (17) by reference to data in the database (11).

2. A method according to claim 1 wherein one of the procedures comprises commuting (25) a projection operation with a fixpoint operation.

3. A method according to claim 1 wherein one of the procedures comprises commuting (27) a selection operation with a fixpoint operation.

4. A method according to claim 3 wherein commuting a selection operation with a fixpoint operation comprises commuting a selection predicate on a direct mapping column (33) with a fixpoint operation.

5. A method according to claim 3 wherein commuting a selection operation with a fixpoint operation comprises commuting a global selection predicate (35) with a fixpoint operation.

6. A method according to claim 3 wherein commuting a selection operation with a fixpoint operation comprises commuting a selection predicate including a join (37) with a fixpoint operation.

7. A method according to claim 1 wherein one of the procedures comprises distributing a join operation (29) over a fixpoint operation.

8. A method according to claim 1 wherein one of the procedures comprises regrouping a join operation (31) and a fixpoint operation.

9. A method according to claim 8 wherein regrouping comprises commutation (39).

10. A method according to claim 8 wherein regrouping comprises association (41).

11. A method of evaluating a recursive query of a computerized database (11), the method comprising: in a computer, translating (13) a recursive query provided by a user into an expression that includes a fixpoint operator; automatically optimizing (15) the expression in the computer according to a set of transformation procedures (43) stored in the computer; and automatically evaluating (17) the optimized expression by reference to data in the computer database.

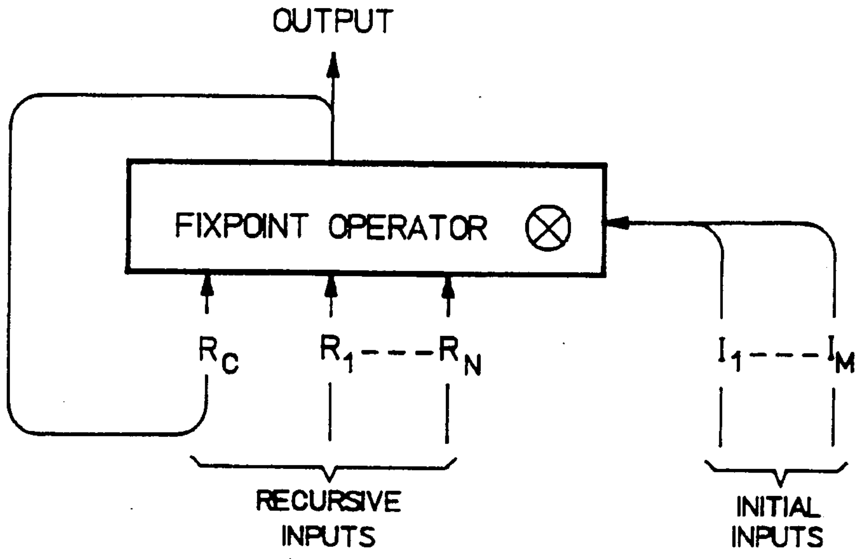

OUTPUT
FIXPOINT OPERATOR
$\otimes$
R_C
R_1---R_N
I_1---I_M
RECURSIVE INPUTS
INITIAL INPUTS

FIG 2

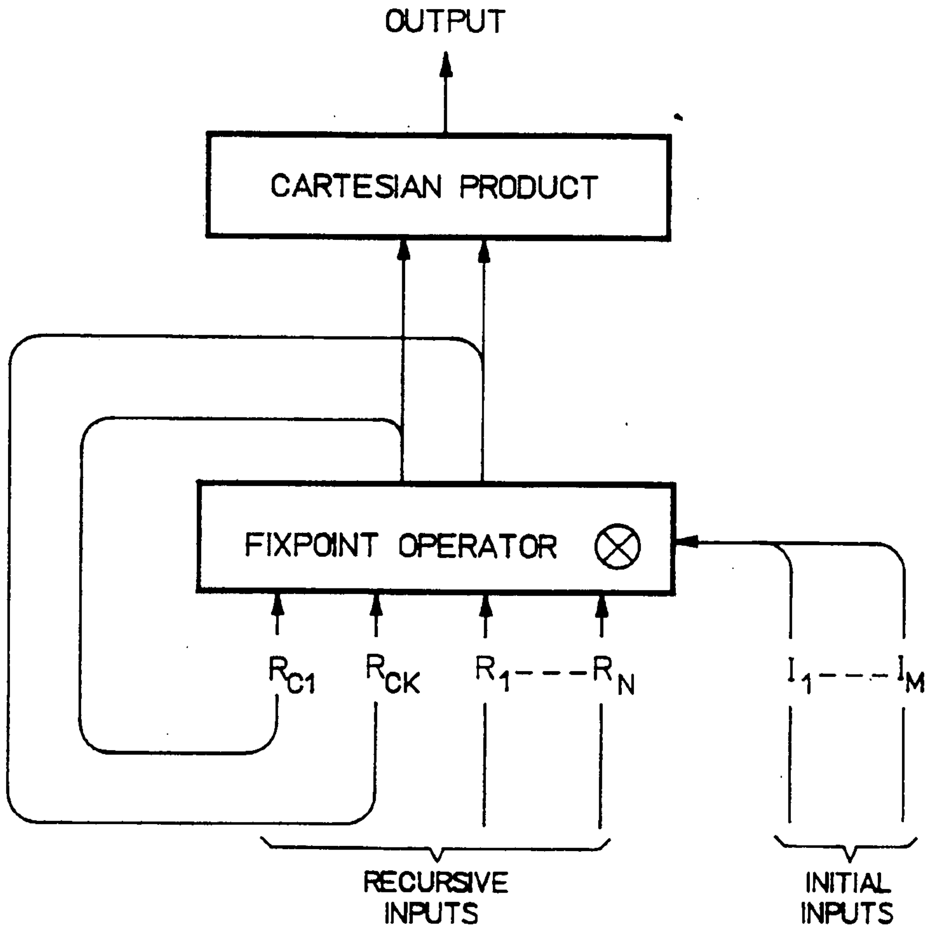

OUTPUT
CARTESIAN PRODUCT
FIXPOINT OPERATOR
$\otimes$
R_{C1}
R_{CK}
R_1---R_N
I_1---I_M
RECURSIVE INPUTS
INITIAL INPUTS

FIG 3

INTERNATIONAL SEARCH REPORT

International Application No PCT/US 92/01458

I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all)[6]

According to International Patent Classification (IPC) or to both National Classification and IPC

Int.Cl. 5 G06F15/403

II. FIELDS SEARCHED

Minimum Documentation Searched[7]

Classification System	Classification Symbols
Int.Cl. 5	G06F

Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched[8]

III. DOCUMENTS CONSIDERED TO BE RELEVANT[9]

Category °	Citation of Document, [11] with indication, where appropriate, of the relevant passages [12]	Relevant to Claim No.[13]
X	ACM SIGMOD RECORD vol. 15, no. 2, June 1986, LE CHESNAY-CéDEX, FRANCE pages 177 - 186; G. GARDARIN ET AL.: 'Evaluation of database recursive logic programs as recurrent function series' see page 177, column 2, line 33 - page 178, column 1, line 14 ---	1,11

° Special categories of cited documents : [10]

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

IV. CERTIFICATION

Date of the Actual Completion of the International Search 30 JUNE 1992	Date of Mailing of this International Search Report 13.07.92
International Searching Authority EUROPEAN PATENT OFFICE	Signature of Authorized Officer KATERBAU R.E. [illegible]

1